Tutoriel - Programmer avec Processing (2)

TOUTES LES FICHES

par Yves Durasnel

PUBLIC	PARTICIPANTS	ANIMATEURS	NIVEAU	PRÉPARATION	ACTIVITÉ

Description

Approfondir la connaissance de Processing en abordant les programmes en mode interactif ou transactionnel

Objectifs

–

Matériel

1 vidéo projecteur et 1 ordinateur maître
Accès internet non indispensable, processing doit alors être téléchargé et/ou installé au préalable
1 ordinateur par participant ou groupe de 2

Compétences travaillées

Concentration
Précision
Curiosité

Contenus utilisés

–

Pré-requis

Avoir suivi la première séquence “Programmer avec Processing (1)”

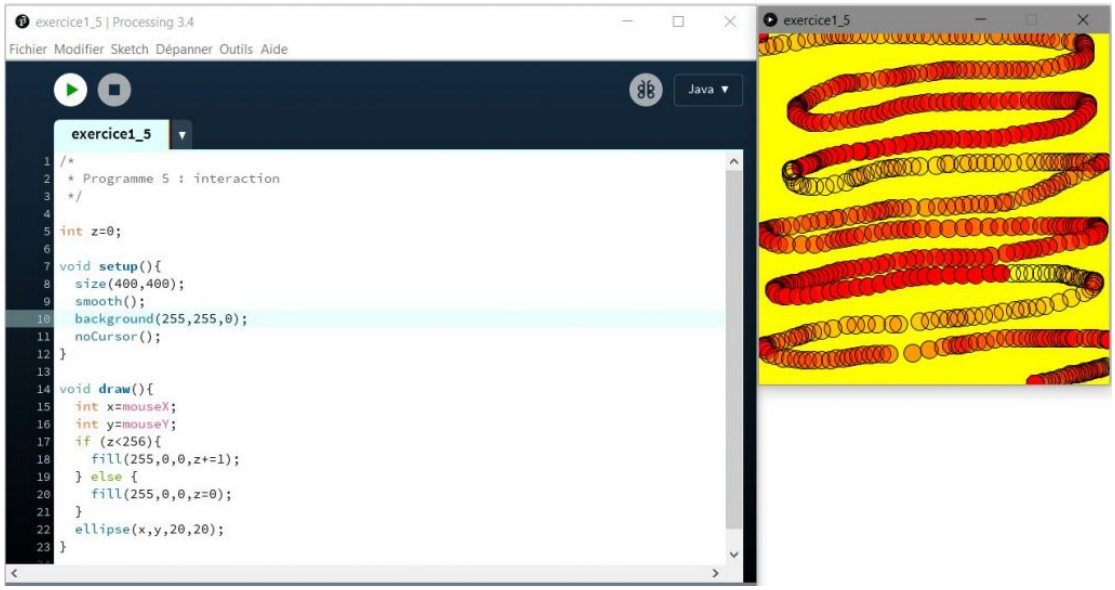
Après les traitements par lots nous allons aborder le mode interactif

Cinquième programme : interaction

Ce code permet de suivre le pointeur de la souris, déplacé par l'utilisateur, et de tracer des cercles le long de la trajectoire de la souris.

Notez que le pointeur et le curseur sont 2 choses différentes même s'ils sont souvent confondus, ici le curseur n'est pas affiché (instruction nocursor()) mais l'application est capable de suivre les mouvements de la souris, c'est à dire du pointeur.

Résultat



Code utilisé :

```
/*
* Programme 5 : interaction
*/
int z=0;
void setup(){
   size(400,400);
   smooth();
   background(255,255,0);
   noCursor();
}
void draw(){
    int x=mouseX;
    int y=mouseY;
```

```
if (z<256){
    fill(255,0,0,z+=1);
} else {
    fill(255,0,0,z=0);
}
ellipse(x,y,20,20);}
```

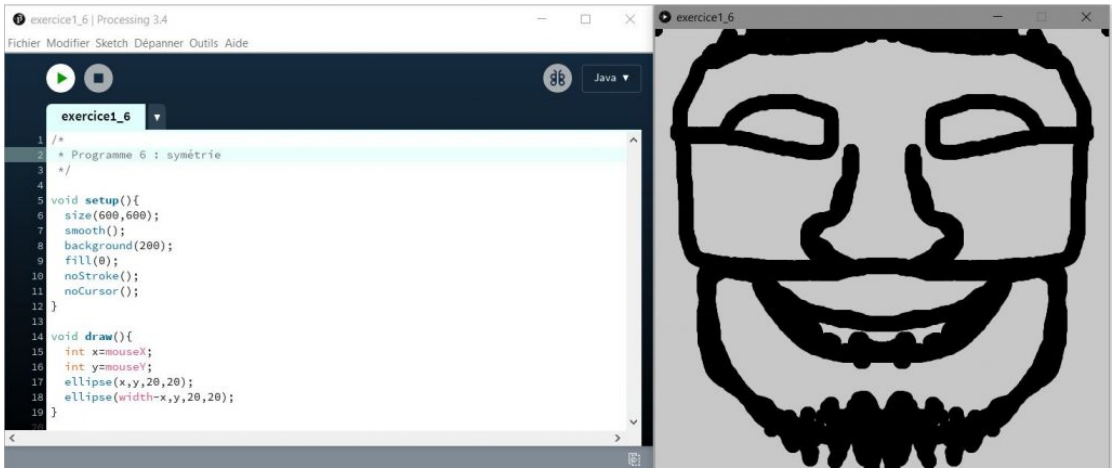
Analyse du code, remarquez :

- Les variables globales mouseX et mouseY contiennent les coordonnées du pointeur de la souris, elles sont définies par le langage Processing.
- La variable z est aussi une variable globale mais définie par le programmeur, elle et initialisée à 0 au début de l'exécution du programme.
- A chaque rafraîchissement d'écran un cercle est tracé à l'emplacement du pointeur, sa transparence varie à chaque rafraichissement d'écran et la couleur rouge du cercle est mélangée au fond jaune pour donner différentes valeurs d'orange.
- Les cercles sont superposés, le dernier cercle tracé apparaît au premier plan et recouvre les précédents.

Sixième programme : symétrie

Deuxième programme permettant de suivre la position du pointeur de la souris. Cette fois avec un effet de symétrie. Le résultat n'est peut-être pas extraordinaire, mais il s'agit simplement d'explorer les pistes ouvertes par les possibilités de la géométrie et des mathématiques rendues simples par le programme Processing.

Résultat



Code utilisé :

```
/*
* Programme 6 : symétrie
*/
```

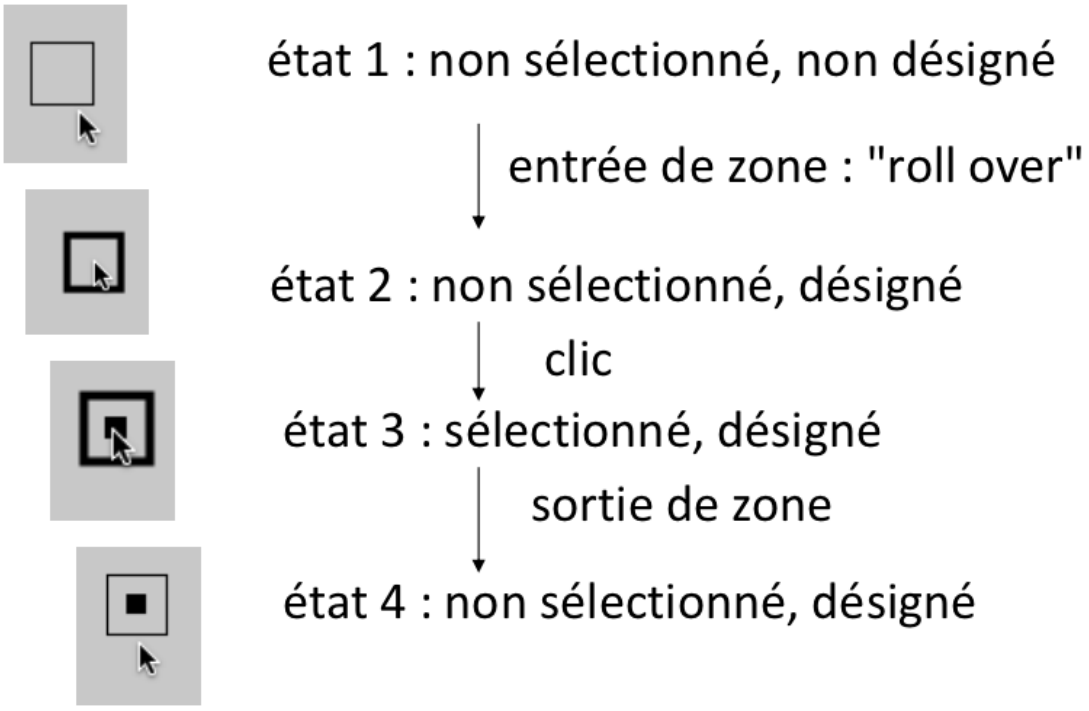
```
void setup(){
   size(600,600);
   smooth();
   background(200);
   fill(0);
   noStroke();
   noCursor();
}
void draw(){
   int x=mouseX;
   int y=mouseY;
   ellipse(x,y,20,20);
   ellipse(width-x,y,20,20);}
```

Analyse du code, remarquez :

- A chaque rafraîchissement d'écran, 2 cercles sont tracés.
- Le premier à la position du curseur de la souris (position x), le deuxième symétriquement par rapport à l'axe vertical central (position width – x).

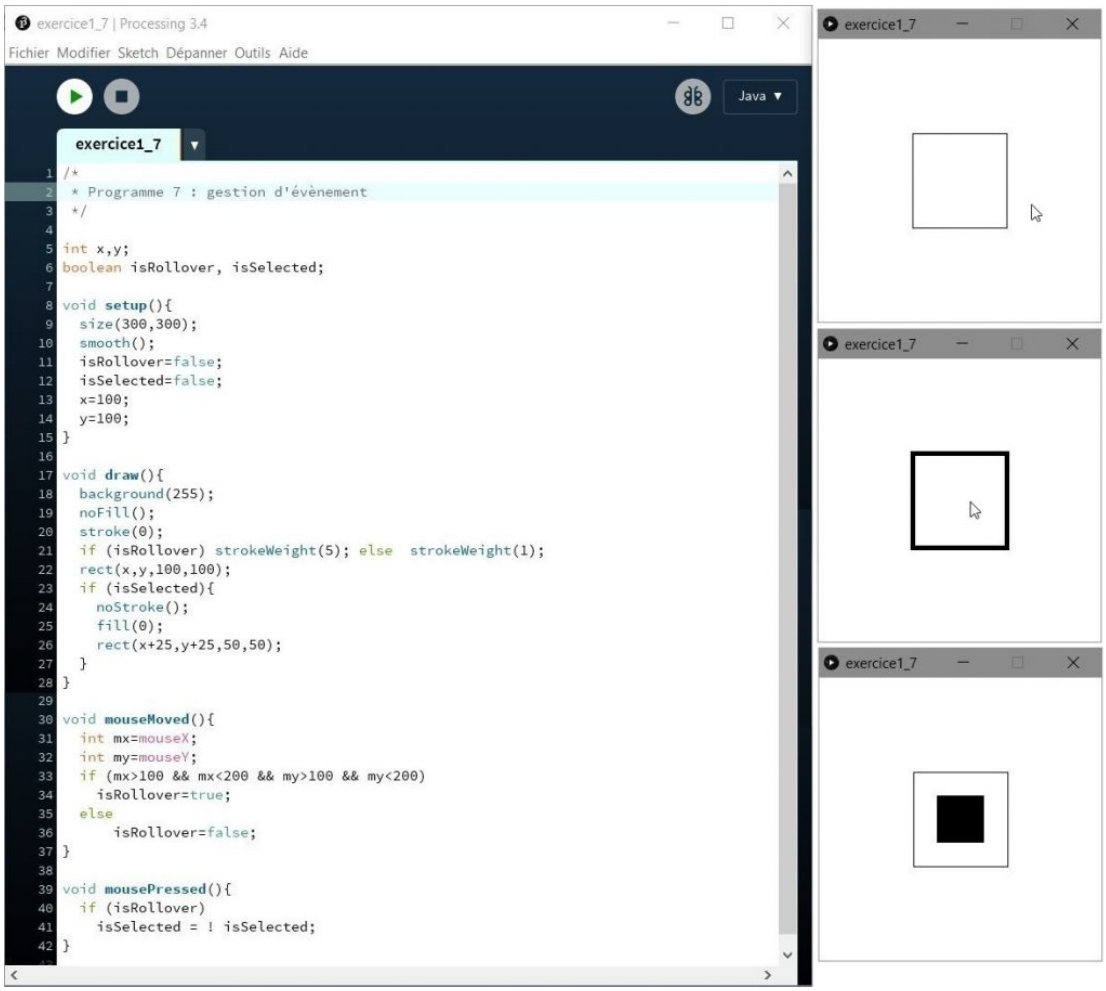
Septième programme : gestion d'événements

Gestion d'un bouton en fonction du survol par la souris ainsi que du clic. Le bouton peut prendre 4 états différents suivant les actions de l'utilisateur.



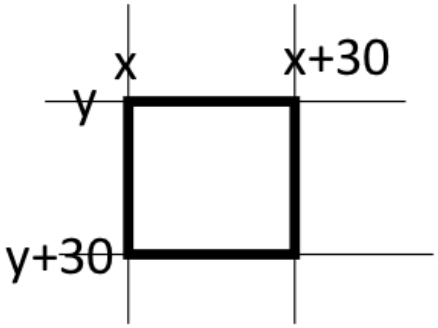
Quelques fonctions utiles pour gérer les événements provoqués par la souris et le clavier :
mousePressed() bouton de souris pressé
mouseReleased() bouton de souris relâché
mouseMoved() souris déplacée
mouseDragged() souris déplacée avec bouton pressé
keyPressed() touche clavier pressée
keyReleased() touche clavier relâchée

Résultat



Détection du rollover

bouton = un carré
en coord (x,y) de largeur
30 pixels




```
si    (mouseX > x) et (mouseX < x+30)
      et (mouseY >y) et (mouseY < y+30)
alors le curseur est dans la boite du bouton
sinon il est dehors
```

Le “roll over” (en français “rouler dessus”) désigne l’action de faire passer le curseur de la souris sur un élément à l’écran.
L’expression conditionnelle qui permet de détecter si le pointeur survole le bouton est explicitée en langage courant pour permettre de comprendre la logique.

2 variables globales booléennes (dont l’état ne peut être que oui ou non) sont initialisées au démarrage du programme :
isRollover pour mémoriser la position du pointeur (est-il à l’intérieur ou l’extérieur du bouton ?)
isSelected pour mémoriser le clic de souris (l’utilisateur a-t-il cliqué ou non sur le bouton ?)

Code utilisé :

```
/*
* Programme 7 : gestion d'évènement
*/
int x,y;
boolean isRollover, isSelected;
void setup(){
   size(300,300);
   smooth();
   isRollover=false;
   isSelected=false;
   x=100;
   y=100;
}
void draw(){
   background(255);
   stroke(0);
   noFill();
   if (isRollover) strokeWeight(5); else strokeWeight(1);
   rect(x,y,100,100);
```

```
   if (isSelected){
       noStroke();
       fill(0);
       rect(x+25,y+25,50,50);
 }
}
void mouseMoved(){
   int mx=mouseX;
   int my=mouseY;
   if (mx>100 && mx<200 && my>100 && my<200)
       isRollover=true;
   else
       isRollover=false;
}
void mousePressed(){
   if (isRollover) isSelected =! isSelected;}
```

Analyse du code, remarquez :

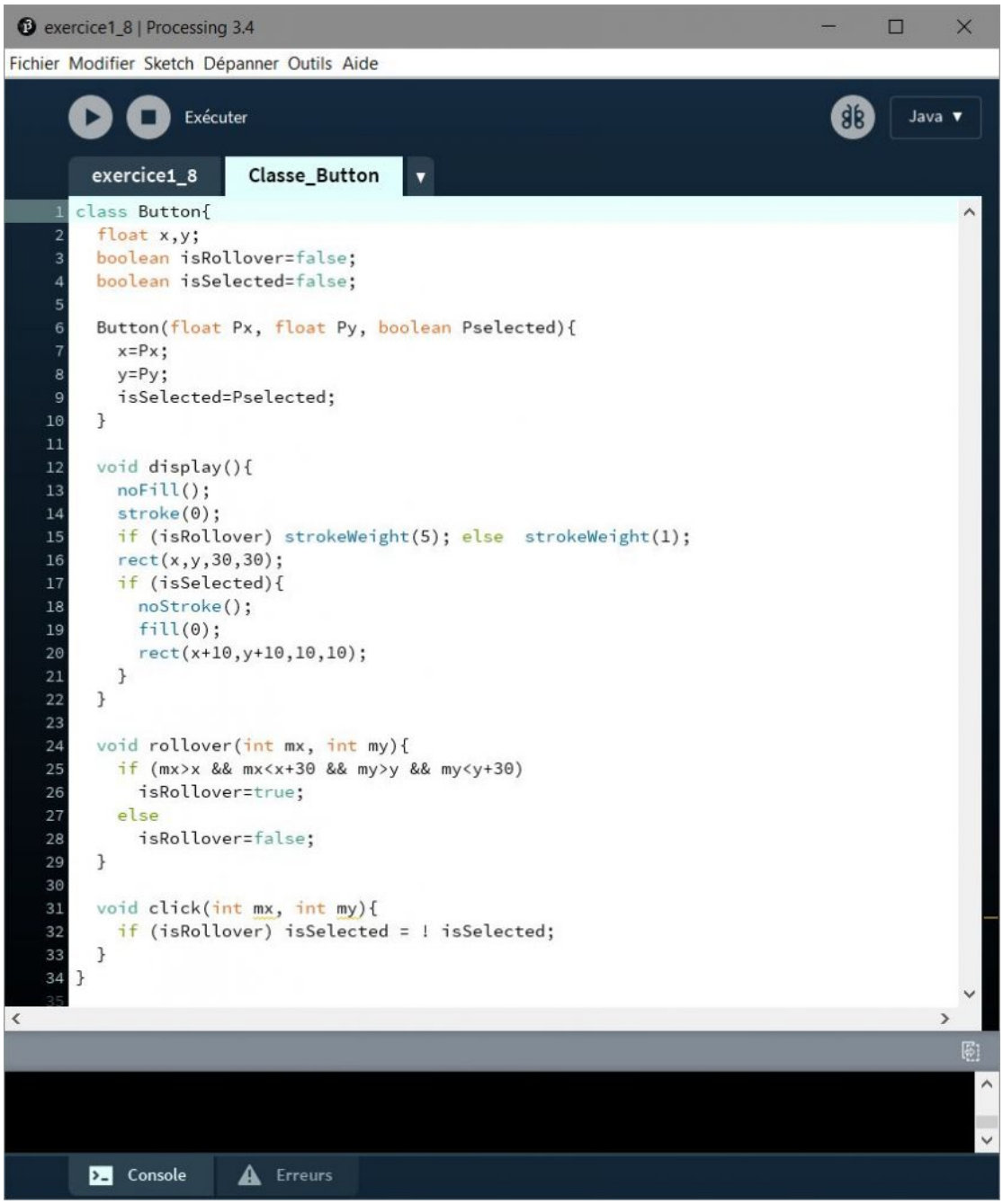
- La construction des structures de test “si” (if en anglais).
- Les fonctions système qui détectent les événement de la souris : mouseMoved() qui détecte si le curseur est à l’intérieur du bouton mousePressed() qui détecte les clics de souris.
- L’état des variables globales isRollover et isSelected pilotées par les fonctions mouseMoved() et mousePressed() qui permettent à la fonction principale draw() d’afficher l’état du bouton en fonction des actions de l’utilisateur.

Huitième programme : approche objet

Comment faire si on veut 2 (ou 100) boutons ? Écrire autant de tests que de bouton ? Modifier les paramètres chaque fois qu’un bouton est déplacé ?

Pour répondre à ces questions nous allons créer une “**classe**” (un modèle de bouton) qui nous permettra d’utiliser autant de boutons que nécessaire, chaque copie (on dit “ **instance**”) ayant un comportement distinct. On dit aussi que le bouton est un “**objet**”.

Onglet de la classe Button



Code de la classe :

```
class Button{
    float x,y;
    boolean isRollover=false;
    boolean isSelected=false;
    Button(float Px, float Py, boolean Pselected){
        x=Px;
```

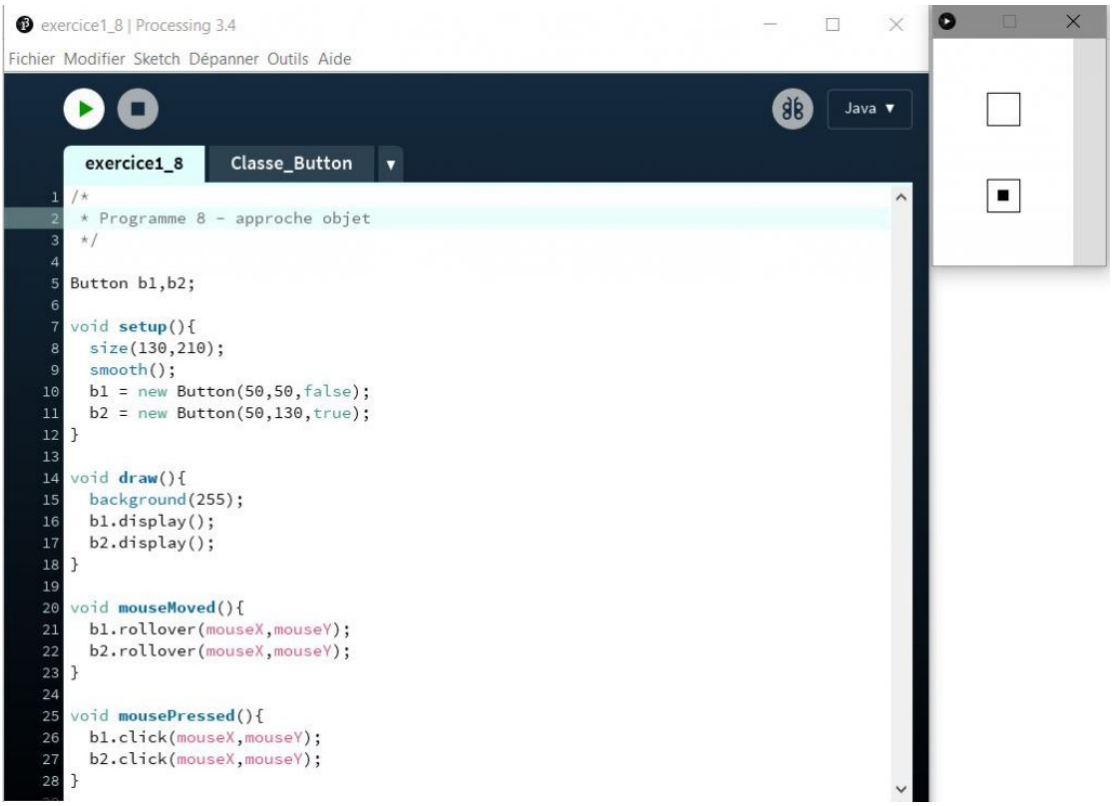
```
    y=Py;
    isSelected=Pselected;
}
void display(){
    noFill();
    stroke(0);
    if (isRollover) strokeWeight(5); else  strokeWeight(1);
    rect(x,y,30,30);
    if (isSelected){
        noStroke();
        fill(0);
        rect(x+10,y+10,10,10);
    }
}
void rollover(int mx, int my){
    if (mx>x && mx<x+30 && my>y && my<y+30)
        isRollover=true;
    else
        isRollover=false;
}
void click(int mx, int my){
    if (isRollover) isSelected =! isSelected;
}}
```

Analyse du code de la classe, remarquez :

- Le code est saisi dans un onglet différent de l'onglet du programme principal, ce n'est pas obligatoire mais plus souple si on désire réutiliser le même code dans un autre programme.
- L'ensemble du code de la classe est contenu entre les accolades suivant la déclaration class Button{ … }. La fonction Button() qui a le même nom que la classe est appelée "**constructeur**" des objets boutons.
- Les fonctions display(), rollover() et click() sont les "**méthodes**" de la classe, leur code est extrait des fonctions draw(), mouseMoved() et mousePressed() du programme 7.
- Les variables x, y, isRollover et isSelected ont une visibilité limitée, chaque instance de bouton utilise ses propres valeurs.

Nos boutons sont les embryons d'une bibliothèque d'interface graphique. Il reste à faire que ces boutons servent à quelque chose (déclenchent des actions…). On pourra également par la suite étendre notre classe en profitant de la notion d'héritage qui permet à partir des objets de base de créer nouveaux objets avec plus de fonctions et une apparence améliorée.

Onglet principal et résultat



Code utilisé :

```
/*
* Programme 8 - approche objet
*/
Button b1,b2;
void setup(){
   size(130,210);
   smooth();
   b1 = new Button(50,50,false);
   b2 = new Button(50,130,true);
}
void draw(){
   background(255);
   b1.display();
   b2.display();
}
void mouseMoved(){
   b1.rollover(mouseX,mouseY);
   b2.rollover(mouseX,mouseY);
}
void mousePressed(){
   b1.click(mouseX,mouseY);
   b2.click(mouseX,mouseY);}
```

Analyse du code, remarquez :

- La déclaration de 2 boutons b1 et b2 de la classe Button.
- Dans la fonction setup() leur instanciation par le mot clé new, chaque bouton ayant une position et un état différent grâce aux paramètres passés à la fonction Button().
- Dans les fonctions draw(), mouseMoved() et mousePressed() la gestion de chacun des boutons en fonction des actions de l'utilisateur.
- Les événements sont communiqués aux boutons au travers de leurs méthodes (b1.rollover par exemple est la méthode rollover du bouton b1, c'est à dire l'action à réaliser quand la souris survole ce bouton).

Conclusion

Le mode interactif ou transactionnel est la base du fonctionnement de l'informatique et de l'internet, il est basé sur la gestion d'événements (clic sur un bouton, saisie d'une adresse mail…). Quand vous utilisez un site de commerce en ligne vous interagissez avec ses serveur pour remplir votre panier, puis vous effectuez une transaction pour payer sur les serveurs de votre banque.

Dans la séquence suivante nous allons mettre en oeuvre ces techniques, nous appuyer sur la programmation objet pour simplifier notre travail et découvrir d'autres outils. Nous allons aborder les fonctionnalités graphiques de Processing pour produire des animations comme au cinéma.

Yves Durasnel
Club sympaTIC
Octobre 2018

Inspiré du FUN Mooc "Introduction aux technologies des médias interactifs numériques"
par Pierre Cubaud et Stéphane Natkin du CNAM

2 Pour aller plus loin

Conseil médiation

Pour aller plus plus loin sur le sujet, nous vous conseillons de vous référer à la fiche outil "Algorithmes et langages de programmation"